

Containment of Nested Regular Expressions

Juan Reutter

Abstract

Nested regular expressions (NREs) have been proposed as a powerful formalism for querying RDFS graphs, but research in a more general graph database context has been scarce, and static analysis results are currently lacking. In this paper we investigate the problem of containment of NREs, and show that it can be solved in PSPACE, i.e., the same complexity as the problem of containment of regular expressions or regular path queries (RPQs).

1 Introduction

Graph-structured data has become pervasive in data-centric applications. Social networks, bioinformatics, astronomic databases, digital libraries, Semantic Web, and linked government data, are only a few examples of applications in which structuring data as graphs is, simply, essential.

Traditional relational query languages do not appropriately cope with the querying problematics raised by graph-structured data. The reason for this is twofold. First, in the context of graph databases one is typically interested in *navigational* queries, i.e. queries that traverse the edges of the graph while checking for the existence of paths satisfying certain conditions. However, most relational query languages, such as SQL, are not designed to deal with this kind of recursive queries [1]. Second, current graph database applications tend to be massive in size (think, for instance, of social networks or astronomic databases, that may store terabytes of information). Thus, one can immediately dismiss any query language that cannot be evaluated in polynomial time (or even in linear time!). But then even the core of the usual relational query languages – *conjunctive* queries (CQs) – does not satisfy this property. In fact, parameterized complexity analysis tells us that – under widely-held complexity theoretical analysis – CQs over graph databases cannot be evaluated in time $O(|G|^c \cdot f(|\varphi|))$, where $c \geq 1$ is a constant and $f : \mathbb{N} \rightarrow \mathbb{N}$ is a computable function [14].

This raises a need for languages that are specific for the graph database context. The most commonly used *core* of these languages are the so-called *regular path queries*, or RPQs [7], that specify the existence of paths between nodes, with the restriction that the labels of such path belong to a regular language. The language of RPQs was later extended with the ability to traverse edge backwards, providing them with a *2-way* functionality. This gives rise to the notion of 2RPQs [5].

Nested regular expressions are a graph database language that aims to extend the possibility of using regular expressions, or 2-way regular expressions, for querying graphs with an *existential test* operator $[(\cdot)]$, also known as *nesting* operator, similar to the one in XPath [10]. This class of expressions was proposed in [15] for querying Semantic Web data, and have received a fair deal of attention in the last years [11, 2, 3].

We say that Here we study the problem of containment of NREs, which is the following problem:

Problem:	NRECONTAINMENT
Input:	NREs Q_1 and Q_2 over Σ .
Question:	Is $Q_1 \subseteq Q_2$?

Note that we study this problem for the restricted case when all the possible input graphs are *semipaths*. The general case will be shown in an extended version of the manuscript.

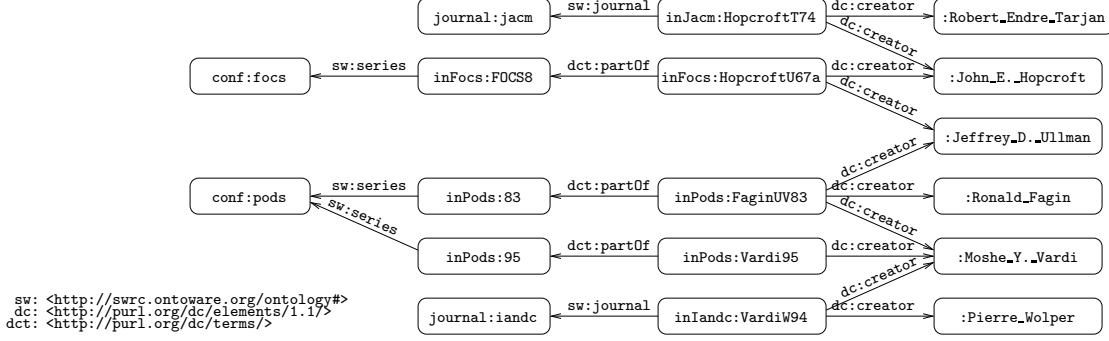


Figure 1: A fragment of the RDF Linked Data representation of DBLP [8] available at <http://dblp.13s.de/d2r/>

2 Preliminaries

2.1 Graph Database and queries

Graph databases. Let \mathbf{V} be a countably infinite set of *node ids*, and Σ a finite alphabet. A *graph database* G over Σ is a pair (V, E) , where V is a finite set of node ids (that is V is a finite subset of \mathbf{V}) and $E \subseteq V \times \Sigma \times V$. That is, G is an *edge-labeled* directed graph, where the fact that (u, a, v) belongs to E means that there is an edge from node u into node v labeled a . For a graph database $G = (V, E)$, we write $(u, a, v) \in G$ whenever $(u, a, v) \in E$.

Nested Regular Expressions.

The language of *nested regular expressions* (NREs) were first proposed in [15] for querying Semantic Web data. Next we formalize the language of nested regular expressions in the context of graph databases.

Let Σ be a finite alphabet. The NREs over Σ extend classical regular expressions with an *existential nesting test* operator $[\cdot]$ (or just nesting operator, for short), and an *inverse* operator a^- , over each $a \in \Sigma$. The syntax of NREs is given by the following grammar:

$$R := \varepsilon \mid a \ (a \in \Sigma) \mid a^- \ (a \in \Sigma) \mid R \cdot R$$

$$R^* \mid R + R \mid [R]$$

As it is customary, we use n^+ as shortcut for $n \cdot n^*$.

Intuitively, NREs specify pairs of node ids in a graph database, subject to the existence of a path satisfying a certain regular condition among them. That is, each NRE R defines a binary relation $\llbracket R \rrbracket_G$ when evaluated over a graph database G . This binary relation is defined inductively as follows, where we assume that a is a symbol in Σ , and n, n_1 and n_2 are arbitrary NREs:

$$\begin{aligned} \llbracket \varepsilon \rrbracket_G &= \{(u, u) \mid u \text{ is a node id in } G\} \\ \llbracket a \rrbracket_G &= \{(u, v) \mid (u, a, v) \in G\} \\ \llbracket a^- \rrbracket_G &= \{(u, v) \mid (v, a, u) \in G\} \\ \llbracket n_1 \cdot n_2 \rrbracket_G &= \llbracket n_1 \rrbracket_G \circ \llbracket n_2 \rrbracket_G \\ \llbracket n_1 + n_2 \rrbracket_G &= \llbracket n_1 \rrbracket_G \cup \llbracket n_2 \rrbracket_G \\ \llbracket n^* \rrbracket_G &= \llbracket \varepsilon \rrbracket_G \cup \llbracket n \rrbracket_G \cup \llbracket n \cdot n \rrbracket_G \cup \llbracket n \cdot n \cdot n \rrbracket_G \cup \dots \\ \llbracket [n] \rrbracket_G &= \{(u, u) \mid \text{there exists } v \text{ s.t. } (u, v) \in \llbracket n \rrbracket_G\}. \end{aligned}$$

Here, the symbol \circ denotes the usual composition of binary relations, that is, $\llbracket n_1 \rrbracket_G \circ \llbracket n_2 \rrbracket_G = \{(u, v) \mid \text{there exists } w \text{ s.t. } (u, w) \in \llbracket n_1 \rrbracket_G \text{ and } (w, v) \in \llbracket n_2 \rrbracket_G\}$.

Example 2.1 Let G_1 be the graph database in Figure 1. The following is a simple NRE that matches all pairs (x, y) such that x is an author that published a paper in conference y :

$$n_1 = \text{creator}^- \cdot \text{partOf} \cdot \text{series}$$

For example the pairs $(:\text{Jeffrey.D. Ullman}, \text{conf:focs})$ and $(:\text{Ronald Fagin}, \text{conf:pods})$ are in $\llbracket n_1 \rrbracket_G$. Consider now the following expression that matches pairs (x, y) such that x and y are connected by a coauthorship sequence:

$$n_2 = (\text{creator}^- \cdot \text{creator})^+$$

For example the pair $(:\text{John.E. Hopcroft}, :\text{Pierre Wolper})$, is in $\llbracket n_2 \rrbracket_G$. Finally the following expression matches all pairs (x, y) such that x and y are connected by a coauthorship sequence that only considers conference papers:

$$n_3 = (\text{creator}^- \cdot [\text{partOf} \cdot \text{series}] \cdot \text{creator})^+$$

Let us give the intuition of the evaluation of this expression. Assume that we start at node u . The (inverse) edge creator^- makes us to navigate from u to a paper v created by u . Then the existential test $[\text{partOf} \cdot \text{series}]$ is used to check that from v we can navigate to a conference (and thus, v is a conference paper). Finally, we follow edge creator from v to an author w of v . The $(\cdot)^+$ over the expression allows us to repeat this sequence several times. For instance, $(:\text{John.E. Hopcroft}, :\text{Moshe.Y. Vardi})$ is in $\llbracket n_3 \rrbracket_G$, but $(:\text{John.E. Hopcroft}, :\text{Pierre Wolper})$ is not in $\llbracket n_3 \rrbracket_G$.

Complexity and expressiveness of NREs The following result, proved in [15], shows a remarkable property of NREs. It states that the query evaluation problem for NREs is not only polynomial in *combined* complexity (i.e. when both the database and the query are given as input), but also that it can be solved linearly in both the size of the database and the expression. Given a graph database G and an NRE R , we use $|G|$ to denote the size of G (in terms of the number of edges $(u, a, v) \in G$), and $|R|$ to denote the size of R .

Proposition 2.2 (from [15]) *Checking, given a graph database G , a pair of nodes (u, v) , and an NRE R , whether $(u, v) \in \llbracket R \rrbracket_G$, can be done in time $O(|G| \cdot |R|)$.*

On the expressiveness side, NREs subsume several important query languages for graph databases. For instance, by disallowing the inverse operator a^- and the nesting operator $[\cdot]$ we obtain the class of *regular path queries* (RPQs) [7, 13], while by only disallowing the nesting operator $[\cdot]$ we obtain the class of RPQs with *inverse* or 2RPQs [5]. (In particular, both expressions n_1 and n_2 in Example 2.1 are 2RPQs). In turn, NREs allow for an important increase in expressive power over those languages. For example, it can be shown that NRE expression n_3 in Example 2.1 cannot be expressed without the nesting operator $[\cdot]$, and hence it is not expressible in the language of 2RPQs (c.f. [15]).

On the other hand, the class of NREs fails capturing more expressive languages for graph-structured data that combine navigational properties with quantification over node ids. Some of the most paradigmatic examples of such languages are the classes of *conjunctive* RPQs and 2RPQs, that close RPQs and 2RPQs, respectively, under conjunctions and existential quantification. Both classes of queries have been studied in depth, as they allow identifying complex patterns over graph-structured data [6, 9, 4].

3 Containment of NREs over paths

3.1 Problem Definition

Let us begin with some notation.

Along the proof we assume that Σ includes all *reverse symbols*. More precisely, if Σ' is an alphabet, we work instead with the alphabet $\Sigma = \Sigma' \cup \{a^- \mid a \in \Sigma'\}$. Let $G = (V, E)$ be a graph over Σ . A semipath in G is a sequence $u_1, a_1, u_2, a_2, \dots, u_m, a_m, u_{m+1}$, where each u_i belongs to V , each a_i belongs to Σ , and

for each u_i, a_i, u_{i+1} , we have that (u_i, a_i, u_{i+1}) belongs to E , if a_i is not a reverse symbol, and (u_{i+1}, a_i, u_i) belongs to E if a_i is a reverse symbol, i.e., of form a^- for some $a \in \Sigma'$. A semipath is simple if all of its nodes are distinct. Finally, a graph G *resembles a (simple) semipath* if there is a (simple) semipath π in G of the form above such that the nodes of G are precisely $\{u_1, \dots, u_n\}$ and the edges of G are precisely those that witness the above definition.

As we have mentioned, we study NRECONTAINMENT only when the input graphs are semipaths. We are now ready to describe our goal which is to show that the following problem is in PSPACE: Given NREs Q_1 and Q_2 over Σ , decide whether $\llbracket Q_1 \rrbracket_G \subseteq \llbracket Q_2 \rrbracket_G$, for all graphs G over Σ such that G resembles a simple semipath. In what follows, we refer to this problem as SP-NRECONTAINMENT.

3.2 Alternating 2-way finite automata

Following [12], an *Alternating 2-way finite automaton*, or A2FA for short, is a tuple $A = (Q, q_0, U, F, \Sigma, \delta)$, where Q is the set of states, $U \subseteq Q$ is a set of *universal* states, q_0 is the initial state, $F \subseteq Q$ is the set of final states, Σ is the input alphabet (we also use symbols % and \$ not in Σ as the start and end markers of the string), and the transition function is $\delta : Q \times (\Sigma \cup \{\%, \$\}) \rightarrow 2^{Q \times \{-1, 0, 1\}}$.

The numbers $-1, 0, 1$ in the transition stand for moving back, staying and moving forward, respectively. The input is delimited with % at the beginning and \$ at the end. For convenience, we assume that the automaton starts in state q_0 while reading the symbol \$ of the string. Finally, we impose the following restriction on δ : the machine can only branch into an existential or universal state while the automaton is not moving backwards or forwards. That is, If for some states p, q and symbol $a \in \Sigma$ we have that $(p, -1)$ or $(p, 1)$ belong to $\delta(q, a)$, then it must be the case that $\delta(q, a) = \{(p, -1)\}$ or $\{(p, 1)\}$. Obviously we do not lose expressive power while imposing this restriction, and we can still simulate any non-deterministic two way automaton with A2FAs.

Semantics Semantics are given in terms of computation trees over instantaneous descriptions. An *instantaneous description* (ID) is a triple of form (q, w, i) , where q is a state, w is a word in $\% \Sigma^* (\epsilon \| \$)$ and $1 \leq i \leq |w| + 1$. Intuitively, it represents the state of the current computation, the string it has already read, and the current position of the automata. An ID is *universal* if $q \in U$ and *existential* otherwise, and accepting IDs are of form $(q, w, |w| + 1)$ for $w \in \% \Sigma^* \$$ and $q \in F$.

Let $w = a_1, \dots, a_n$, for each $a_i \in \Sigma \cup \{\%, \$\}$. The transition relation \Rightarrow is defined as follows:

- $(q, w, i) \Rightarrow (p, w, i)$, if $(p, 0) \in \delta(q, a_i)$ and $1 \leq i \leq n$;
- $(q, w, i) \Rightarrow (p, w, i + 1)$, if $(p, 1) \in \delta(q, a_i)$ and $1 \leq i \leq n$; and
- $(q, w, i) \Rightarrow (p, w, i - 1)$, if $(p, -1) \in \delta(q, a_i)$ and $1 < i \leq n$.

A *computation tree* Π of an A2FA $A = (Q, q_0, U, F, \Sigma, \delta)$ is a finite, nonempty tree with each of its nodes π labelled with an ID $l(\pi)$, and such that

1. If π is a non-leaf node and $l(\pi)$ is universal, let I_1, \dots, I_k be all IDs such that $l(\pi) \Rightarrow I_j$ for each $1 \leq j \leq k$. Then π has exactly k children π_1, \dots, π_k , where $l(\pi_j) = I_j$; and
2. If π is a non leaf node and $l(\pi)$ is existential, then π has exactly one child π' such that $l(\pi) \Rightarrow l(\pi')$.

Finally, an *accepting computation tree* of A over w is a computation tree Π whose root is labelled with $(q_0, \%w$, $|w| + 2)$ and each of its leaves are labelled with an accepting ID.$

We need the following theorem. It follows immediately from the results in [12]:

Proposition 3.1 *Given a A2FA A , it is PSPACE-complete to decide whether the language of A is empty.*

3.3 Proof of SP-NRECONTAINMENT

The idea is to code acceptance of strings by NREs using alternating 2-way automata. More precisely, given an NRE R , we construct an A2FA A_R such that the language of A_R corresponds, in a precise sense, to all those words w such that $\llbracket R \rrbracket_G$ is nonempty for all those graphs G that resemble the simple semipath w .

Construction of A_R . We define the translation by induction, all states are existential unless otherwise noted. Along the construction, we shall be *marking*, in each step, a particular state of the automata. We use this mark in the construction. Furthermore, for the sake of readability we shall drop the assumption that these automaton are deterministic when moving the head forward or backwards.

- If $R = a$, then $A_R = (\Sigma, \{q_0, q_f, q_r\}, \emptyset, q_0, \delta, q_f)$, with δ defined as:

$$\begin{aligned}\delta(q_0, a) &= \{(q_f, 1), (q_r, -1)\} \\ \delta(q_0, b) &= \{(q_r, -1)\}, \text{ for each } b \in \Sigma, b \neq a \\ \delta(q_r, a^-) &= \{(q_f, 0), \}\end{aligned}$$

State q_r and the two way functionality is added so that the automaton correctly accepts when the input is a word of form $\Sigma^* a^- \Sigma^*$ (See [5] for a thorough explanation of this machinery). Moreover, state q_f is *marked*.

- Similarly, if $R = a^-$, then $A_R = (\Sigma, \{q_0, q_f, q_r\}, \emptyset, q_0, \delta, q_f)$, with δ defined as:

$$\begin{aligned}\delta(q_0, a^-) &= \{(q_f, 1), (q_r, -1)\} \\ \delta(q_0, b) &= \{(q_r, -1)\}, \text{ for each } b \in \Sigma, b \neq a^- \\ \delta(q_r, a) &= \{(q_f, 0), \}\end{aligned}$$

State q_f is *marked*.

- Case when $R = R_1 + R_2$. Let $A_{exp_i} = (\Sigma, Q^i, U^i, q_0^i, \delta^i, F^i)$, for $i = 1, 2$, and assume that q_m^i is the marked state from A_{exp_i} . Define $A_R = (\Sigma, Q, U, q_0, \delta, F)$, where $Q = \{q_0, q_f\} \cup Q^1 \cup Q^2$, $U = U^1 \cup U^2$, $F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \cup (F^2 \setminus \{q_m^2\})$ and $\delta = \delta^1 \cup \delta^2$, plus transitions

$$\begin{aligned}\delta(q_0, \varepsilon) &= \{(q_0^1, 0), (q_0^2, 0)\} \\ \delta(q_m^1, \varepsilon) &= \{(q_f, 0)\} \\ \delta(q_m^2, \varepsilon) &= \{(q_f, 0)\}\end{aligned}$$

For each $i = 1, 2$, remove all marks from A_{R_i} , and *mark* state q_f .

- In the case that $R = R_1 \cdot R_2$, let $A_{exp_i} = (\Sigma, Q^i, U^i, q_0^i, \delta^i, F^i)$, for $i = 1, 2$, and assume that q_m^i is the marked state from A_{exp_i} . For each $i = 1, 2$, remove all colorings from A_{R_i} , remove states q_0^i and q_b^i from Q^i and remove from δ^i all transitions mentioning q_0^i or q_g^i . Define $A_R = (\Sigma, Q, U, q_0, \delta, F)$, where $Q = \{q_0, q_f\} \cup Q^1 \cup Q^2$, $U = U^1 \cup U^2$, $F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \cup (F^2 \setminus \{q_m^2\})$ and $\delta = \delta^1 \cup \delta^2$, plus transitions

$$\begin{aligned}\delta(q_0, \varepsilon) &= \{(q_0^1, 0)\} \\ \delta(q_m^1, \varepsilon) &= \{(q_0^2, 0)\} \\ \delta(q_m^2, \varepsilon) &= \{(q_f, 0)\}\end{aligned}$$

For each $i = 1, 2$, remove all marks from A_{R_i} , and *mark* state q_f .

- For $R = R_1^*$, let $A_{exp_1} = (\Sigma, Q^1, U^1, q_0^1, \delta^1, F^1)$, and assume that q_m^1 is the marked state from A_{exp_1} .

Define $A_R = (\Sigma, Q, U^1, q_0, \delta, F)$, where $Q = \{q_0, q_f\} \cup Q^1$, $F = \{q_f\} \cup (F^1 \setminus \{q_m^1\})$ and $\delta = \delta^1$ plus transitions

$$\begin{aligned}\delta(q_0, \varepsilon) &= \{(q_0^1, 0)\} \\ \delta(q_0^1, \varepsilon) &= \{(q_f, 0)\} \\ \delta(q_m^1, \varepsilon) &= \{(q_f, 0), (q_0^1, 0)\}\end{aligned}$$

Remove all marks from A_{R_1} , and *mark* state q_f .

- When $R = [R_1]$, let $A_{exp_1} = (\Sigma, Q^1, U^1, q_0^1, \delta^1, F^1)$, and assume that q_m^1 is the marked state from A_{exp_1} . Then $A_R = (\Sigma, Q, U^1, q_0, \delta, F)$, where $Q = \{q_0, p, q_2, q_f\} \cup Q^1$, $U = U^1 \cup \{p\}$, $F = \{q_f\} \cup F^1$ and $\delta = \delta^1$, plus transitions

$$\begin{aligned}\delta(q_0, \varepsilon) &= \{(p, 0)\} \\ \delta(p, \varepsilon) &= \{(q_f, 0), (q_i^1, 0)\} \text{ (recall that } p \text{ is a universal state)} \\ \delta(q_m^1, a) &= \{(q_m^1, 1)\} \text{ for each } a \in \Sigma\end{aligned}$$

Remove all marks from A_{R_1} , and *mark* state q_f .

Let $A_R = (Q, q_0, U, F, \Sigma, \delta)$ be as constructed by this algorithm. To finish our construction we need to allow A_R to (non deterministically) move backwards from the end of the word, until it reaches a suitable starting point for the computation, and allow every final state to reach the end of the word in its computation. Formally, we define $A'_R = (Q \cup \{q'_0\}, q'_0, U, F, \Sigma \cup \{\$, \delta'),$ where δ' contains all transitions in δ plus transitions $\delta(q'_0, a) = \{(q_0, 0), (q'_0, -1)\}$ for each $a \in \Sigma \cup \{\$\}$ and $\delta(q_f, a) = (q_f, 1)$ for each $a \in \Sigma$.

Notice that the above construction can be computed in polynomial time with respect to R . Furthermore, let q_m be the marked state of A'_R . From its construction, it is clear that every accepting computation tree Π of A_R on input w will have the following form: (1) For some $1 \leq i \leq |w|$ there is a single path from the root to a node π_s such that $l(\pi) = (q_0, w, i)$ and no ancestor of π is labelled with an ID using a state different from q'_0 ; and (2) there is some $1 \leq j \leq |w|$ such that there is exactly one path of nodes $\pi_f, \pi'_f, \pi''_f, \dots$ labelled with $(q_m, w, j), (q_m, w, j+1), \dots, (q_m, w, |w|+1)$. Property (1) represents the automaton searching for its starting point, and (2) represents the end of the computation of the part of A'_R that is representing the non-nesting part of R . We denote such nodes π_s and π_f as the *tacit start* and *tacit ending* of Π . With this definitions we can show the following.

Lemma 3.2 *Let S be a graph over Σ that is a semipath, w the label of the path S , and R a NRE. Then a pair (u_i, u_j) belongs to $\llbracket R \rrbracket_S$ if and only if there is an accepting computation tree of A'_R on input w whose tacit start is labelled with (q_0, w, i) and whose tacit ending is labelled with (q_m, w, j) .*

Proof: Let S be the semipath $u_1, a_1, u_2, a_2, \dots, u_m, a_m, u_{m+1}$, and therefore $w = a_1 \cdot \dots \cdot a_m$, and let $A'_R = (\Sigma, Q, U, q'_0, \delta', F)$ constructed as explained above.

For the **only if direction**, assume that $\llbracket R \rrbracket_S$ contains the pair (u_i, u_j) , $1 \leq i, j \leq m+1$. We prove the above statement by induction on R .

- If $R = a$, for some $a \in \Sigma$, and $(u_i, u_j) \in \llbracket R \rrbracket_S$, then either $j = i+1$ and the edge (u_i, a, u_j) is in S , or $j = i-1$ and the edge (u_j, a^-, u_i) is in S . In the former case the existence of a computation tree is obvious, for the latter case observe that one could use the transitions $(q_0, w, i) \Rightarrow (q_r, w, i-1)$, and then since (u_j, a^-, u_i) is in S we follow transition $(q_r, w, i-1) \Rightarrow (q_f, w, i-1)$.
- Case for $R = a^-$ is analogous to the previous one
- If $R = R_1 + R_2$ and $(u_i, u_j) \in \llbracket R \rrbracket_S$, then $(u_i, u_j) \in \llbracket R_k \rrbracket_S$ for $k = 1$ or $k = 2$, which entails a proper accepting computation tree for A_{R_1} (A_{R_2}) on input w . The statement follows immediately from the construction of A_R .

- If $R = R_1 \cdot R_2$ and $(u_i, u_j) \in \llbracket R \rrbracket_S$, then there is a node u_k of S such that $(u_i, u_k) \in \llbracket R_1 \rrbracket_S$ and $(u_k, u_j) \in \llbracket R_2 \rrbracket_S$. Assume that the initial and marked nodes of A_{R_1} and A_{R_2} are q_0^1, q_m^1 and q_0^2, q_m^2 , respectively. From the induction hypothesis we have that there are accepting computation trees for A_{R_1} and A_{R_2} whose tacit starts are (q_0^1, w, i) and (q_0^2, w, k) , respectively, and the tacit ending of the first tree is labelled with (q_m^1, w, k) . Since A'_R has, by construction, the pair $(q_0^2, 0)$ in $\delta(q_m^2, \varepsilon)$, we can cut the first tree in its tacit ending and plug in the computation tree for A_{R_2} , starting from its tacit start which proves the statement.
- The case when $R = R_1^*$ goes along the same lines as the concatenation, except this time we may have to plug in a greater number of computation trees.
- Finally, if $R = [R_1]$ and $(u_i, u_j) \in \llbracket R \rrbracket_S$, then $u_i = u_j$, and there is some u_k such that $(u_i, u_k) \in \llbracket R_1 \rrbracket_S$. Let q_p be the universal state in A'_R that is not in A_{R_1} . Then the only transitions associated to q_p are $\delta(q_p, \varepsilon) = \{(q_0^1, 0), (q_f, 0)\}$, with q_f being the only marked (final) state of A_R . Our accepting computation tree for A'_R has a path from the root to the tacit start, then a node labeled (q_p, w, i) with children (q_0^1, w, i) and (q_f, w, i) , with the computation tree for A_{R_1} (starting from its tacit start) plugged into the first of these children.

For the **if direction**, assume that there is an accepting computation tree of A_R on input w whose tacit start is labelled with (q_0, w, i) and with its tacit ending labelled with (q_m, w, j) . We now prove that (u_i, u_j) belong to $\llbracket R \rrbracket_S$. The proof is again by induction

- For the base case when $R = a$ (proof for $R = a^-$ is analogous), there are two options for an accepting computation of A_R . Either it is of form $(q_0, w, i) \Rightarrow (q_f, w, i + 1)$, in which case $j = i + 1$ and $a_i = a$, or it is of form $(q_0, w, i) \Rightarrow (q_r, w, i - 1) \Rightarrow (q_f, w, i - 1)$, in which case $j = i - 1$ and $a_i = a^-$. For both cases we obtain that $(u_i, u_j) \in \llbracket R \rrbracket_S$.
- When $R = R_1 + R_2$, by the construction of A_R , any computation tree of A_R can be pruned from its tacit start to obtain a computation tree for one of A_{R_1} or A_{R_2} , from where the statement easily follows.
- When $R = R_1 \cdot R_2$, we can similarly obtain computation trees for A_{R_1} and A_{R_2} , and then conclude that (u_i, u_j) belong to $\llbracket R \rrbracket_S$. Same hold when $R = R_1^*$, except in this case we obtain multiple computation trees for R_1 .
- Finally, if $R = [R_1]$ and there is an accepting computation tree of A_R on input w whose tacit start is labelled with (q_0, w, i) and with its tacit ending labelled with (q_m, w, j) , from the construction of A_R the top part of the computation tree is of form $(q_0, w, i) \Rightarrow (q_p, w, i) \Rightarrow (q_0^1, w, i), (q_m, w, i)$, where q_p is the only universal state of A_R not in A_{R_1} , and q_0^1 is the initial state of A_{R_1} . Then the part of the computation tree that follows from node (q_0^1, w, i) comprises a computation tree for A_{R_1} , i.e., there is a u_k such that $(u_i, u_k) \in \llbracket R_1 \rrbracket_S$. This entails that $(u_i, u_i) \in \llbracket R \rrbracket_S$.

□

Proof for containment For our algorithm of containment, we need to be a little more careful, since for a word w accepted by A_R it is not necessarily the case that u_i and u_j are the start and finish nodes of the semipath S . Thus, we have to distinguish the start/end of the word with the actual piece that is framed by nodes u_i and u_j in the semipath. In order to do that, we augment Σ with two extra symbols S, E . Furthermore, if $A_R = (\Sigma, Q, U, q_0', \delta, F)$, and q_m is the marked state of A_R , we construct $A_R^{S,E} = (\Sigma \cup \{S, E\}, Q \cup \{q_0^S, q_f^E\}, U, q_0^S, \delta^{S,E}, (F \setminus \{q_m\}) \cup \{q_f^E\})$, where δ^* is defined as follows: for each state $q \in Q \setminus U$, we add the pair $(q, 1)$ to $\delta(q, S)$ and $\delta(q, E)$, if q is not q_0 or q_m , the pair $(q_f^E, 1)$ to $\delta(q_m, E)$, $(q_0, 1)$ to $\delta(q_0^S, S)$, plus the pair $(q_0^S, -1)$ to each $\delta(q_0^S, a)$ for $a \in \Sigma \cup \{E\}$.

The intuition is the following. Let R be an NRE and A_R be the A2FA constructed as above. Now assume that there is a semipath $w = u_1, a_1, u_2, \dots, u_n, a_n, u_{n+1}$ and nodes u_i, u_j such that $(u_i, u_j) \in \llbracket R \rrbracket_S$.

By the above Lemma, we have that there is a computation tree for A_R that tacitly starts in (q_0, w, i) and tacitly ends in (q_m, w, j) . The idea of the symbols S and E is to specifically mark the tacit start and end of the piece a_i, \dots, a_{j-1} labeling the semipath between u_i and u_j . Thus, in this case, $A_R^{S,E}$ accepts the word $a_1 \cdots a_{i-1} S a_i \cdots a_{j-1} E a_j \cdots a_n$. It uses intuitively the same computation tree mentioned before, except now it moves backwards in state q_0^S until symbol S is reached, then proceeds with the computation, and the marked branch now ends in q_m^E instead of Q_m , after checking there is a symbol E after a_{j-1} . With this intuition, it is straightforward to show:

Lemma 3.3 *Let $w = u_1, a_1, u_2, \dots, u_n, a_n, u_{n+1}$ be a graph over Σ that is a simple semipath, $w = a_1, \dots, a_n$ the label of the path w , and R a NRE. Then a pair (u_i, u_j) belongs to $\llbracket R \rrbracket_S$ if and only if $A_R^{S,E}$ accepts the word $a_1 \cdots a_{i-1} S a_i \cdots a_{j-1} E a_j \cdots a_n$.*

We can now state our algorithm for solving SP-QUERYCONTAINMENT. On input NREs R_1 and R_2 , we perform the following operations:

1. Compute an NFA $A^{S,E}$ that accepts only those words over $(\Sigma \cup \{S, E\})^*$ of form $w_1 S w_2 E w_3$, for each w_1, w_2, w_3 in Σ^* .
2. Compute $A_{R_1}^{S,E}$ and $A_{R_2}^{S,E}$ as explained above.
3. Compute the A2FA $A^c = (A_{R_2}^{S,E})^c$ whose language is the complement of $A_{R_2}^{S,E}$.
4. Compute the A2FA A whose language is the intersection of the languages $A^{S,E}$, $A_{R_1}^{S,E}$ and A^c .
5. Check that the language of A is empty

We have seen how to perform the second step in polynomial time, and steps (1), (3), (4) can be easily performed in PTIME using standard techniques from automata theory. Finally, Proposition 3.1 shows that step (5) can be performed in PSPACE. Thus, all that is left to prove is that the language of the resulting automata A is empty if and only if $R_1 \subseteq R_2$.

Assume first that $R_1 \subseteq R_2$, and assume for the sake of contradiction that there is a word $w \in L(A)$. We have that w must be of form $a_1 \cdots a_{i-1} S a_i \cdots a_{j-1} E a_j \cdots a_n$, and w is accepted by $A_{R_1}^{S,E}$, but not by $A_{R_2}^{S,E}$. Let S be a graph consisting of the semipath $u_1, a_1, u_2, \dots, u_n, a_n, u_{n+1}$. By Lemma 3.3, nodes $(u_i, u_j) \in \llbracket R_1 \rrbracket_S$, and thus by our assumption (u_i, u_j) must belong to $\llbracket R_2 \rrbracket_S$, but this would imply, again by the lemma, that w is accepted by $A_{R_2}^{S,E}$.

On the other hand if $L(A)$ is empty but $R_1 \not\subseteq R_2$, then for some graph $S = u_1, a_1, u_2, \dots, u_n, a_n, u_{n+1}$ that is a semipath and nodes u_i, u_j it is the case that $(u_i, u_j) \in \llbracket R_1 \rrbracket_S$, yet $(u_i, u_j) \notin \llbracket R_2 \rrbracket_S$. By Lemma 3.3, we have that $w = a_1 \cdots a_{i-1} S a_i \cdots a_{j-1} E a_j \cdots a_n$ is accepted by $A_{R_1}^{S,E}$, and it is not accepted by $A_{R_2}^{S,E}$, thus belonging to A^c . Since clearly w is also in the language of $A^{S,E}$, this means that w belongs to $L(A)$, which is a contradiction.

References

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] P. Barceló, J. Pérez, and J. L. Reutter. Relative expressiveness of nested regular expressions. In *AMW*, pages 180–195, 2012.
- [3] P. Barceló, J. Pérez, and J. L. Reutter. Schema mappings and data exchange for graph databases. In *to appear in ICDT*, page TBD, 2013.
- [4] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Answering regular path queries using views. In *16th International Conference on Data Engineering (ICDE)*, pages 389–398, 2000.

- [5] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Containment of conjunctive regular path queries with inverse. In *7th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 176–185, 2000.
- [6] M. Consens and A. Mendelzon. Graphlog: A visual formalism for real life recursion. In *9th ACM Symposium on Principles of Database Systems (PODS)*, pages 404–416, 1990.
- [7] I. Cruz, A. Mendelzon, and P. Wood. A graphical query language supporting recursion. In *ACM Special Interest Group on Management of Data 1987 Annual Conference (SIGMOD)*, pages 323–330, 1987.
- [8] D2R DBLP bibliography database hosted at L3S research center. <http://dblp.l3s.de/d2r/>, 2013.
- [9] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *17th ACM Symposium on Principles of Database Systems (PODS)*, pages 139–148, 1998.
- [10] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.
- [11] P. Hayes. RDF Semantics, W3C Recommendation. <http://www.w3.org/TR/rdf-mt>, February 2004.
- [12] R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM J. Comput.*, 13(1):135–155, 1984.
- [13] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.
- [14] C. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.
- [15] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.